Cray Productivity Feature Evaluation
Automated Profiling Analysis

1 Automated Profiling Analysis

Automated Profiling Analysis, or APA, is a new feature of the Cray Performance Tools designed to simplify the process of program instrumentation and data collection for purposes of performance analysis.

APA works by first profiling the application for time usage. It uses the profile information gathered to determine an appropriate data collection strategy specifically for that application. APA specifies the strategy in the form of a template file that can be used as input into subsequent performance analysis runs.

1.1 Feature Description

One of the first tasks that a developer tackles when analyzing an application is to determine where the application is spending most of its time. To do this, a timing profile is generated. After this information is available, the developer then wants to understand why time is being spent in certain routines. Typically, the developer will set up a second experiment to collect further information. The APA feature can be used to simplify this set of tasks.

To create a timing profile using the APA feature, the developer simply passes the '-apa' option to the instrumentation utility, pat_build. After the instrumented application executes, APA automatically creates a customized template file for the application that can be used as input to any future performance analyses required for gathering more information. This template file includes a list of all the program's functions, separating out those functions which took small amounts of time so that they will not be included in the next performance analysis. The template file also indicates the kinds of information to collect, such as specifying the hardware counter group to use and listing libraries which should be traced. The user can use this template directly, edit it to refine the next collection of data, or use it to create new experiments for that application.

1.2 Availability for Evaluation

Support for this feature is in all versions of the Cray Performance Analysis Tools Versions 5.0 and higher.

1.3 Benefits

Performance tools provide many options in order to allow the user to control the types of data collected, the volume of data gathered and the best manner of display. The learning curve for using these tools is steep and frequently a developer does not have much time to become familiar with the tools.

APA provides a straightforward set of steps that a developer can follow in order to produce a performance analysis report of an application which focuses on the important aspects of the program, without generating unnecessarily large amounts of data. Following these steps does not require that the developer have a high degree of familiarity with the tools.

In addition, APA creates a template file which can serve as input to a subsequent performance analysis run. This template file can be easily modified to fine tune the data collection on future runs. It is easier and less error-prone for the developer to modify an existing template file that to create a new input from scratch.

1.4 Restrictions

Versions 5.0 and lower of Cray Performance Analysis tools must be used on programs that are statically linked. Support for dynamic libraries will be included in Version 5.1.

Only non-static function entry points at global scope and written in C, C++ or Fortran can be instrumented.

2 Setup and Usage

The following examples here illustrate how to setup your environment and use the APA feature.

2.1 Environment Setup

APA is included in the Cray Performance Tools Releases 5.0 and higher.

The following example illustrates how to check the availability and release level of CrayPat on a Cray XT machine.

```
# load the necessary modules
$ module load xt-craypat

# Check the version using pat_build or pat_report
$ pat_build –V
CrayPat/X: Version 5.0 Revision 2786 08/31/09 12:18:23
```

If the version reported is older than 5.0, contact your system administrator to update the CrayPat package.

2.2 General Use

General usage of this feature is provided in section 2.4 of the manual Using Cray Performance Analysis Tools, S-2376-50. This document can be found on the Cray documentation web site, docs.cray.com.

The example below illustrates how APA can be used.

2.3 Using the Provided Example

2.3.1 Material Location

An electronic copy of the example will be provided along with this feature description. It can also be requested via one of the contacts listed at the end of this document.

2.3.2 Resource Requirement

There are no special resource requirements, other than access to an XT system which runs CNL and is capable of executing your application.

2.3.3 Running the Example

The example used to demonstrate this feature is based on the SWIM Fortran program. The following example code shows how to setup and generate an executable.

```
# load the craypat module
$ module load xt-craypat

# untar the example file
# must be in a lustre filesystem
$ tar –xf apa_example.tar

# change to the src directory
$ cd apa_example/src

# choose makefile for the compilers at your site
$ module load PrgEnv-cray
$ ln –s makefile makefile.cray
# or
$ module load PrgEnv-pgi
$ ln –s makefile makefile.pgi

# create the swim executable
$ make
```

The next set of steps guides you through using APA to instrument and examine the code. The example includes a script file which contains a version of these commands that uses variables for most of the program and file names. You can either type the commands in the example box below or use the supplied script.

```
# Start in the bin directory
cd apa_example/bin

# Step 1. Instrument the original program
export PAT_RT_EXPFILE_NAME=swim
pat_build -O apa swim

# Step 2. Run the instrumented code (may need to use batch)
aprun -n 4 swim+pat

# Step 3. Use pat_report to process the data file
pat_report_filename=$PAT_RT_EXPFILE_NAME.xf
pat_report $pat_report_filename > pat_report.sampling.stdout
mv $PAT_RT_EXPFILE_NAME.ap2 $PAT_RT_EXPFILE_NAME.sampling.ap2

# Step 4. Reinstrument the program
pat_apa_filename=$PAT_RT_EXPFILE_NAME.apa
pat_build -O $pat_apa_filename

# Step 5. Run the reinstrumented program
export PAT_RT_EXPFILE_REPLACE=1
aprun -n 4 swim+apa

# Step 6. Pat_report
pat_report $pat_report_filename > pat_report.apa.stdout
```

2.3.4 Using Your Own Application

Using the above example as a guide, try this on your own application. A good candidate code will use C and/or Fortran and perform some combination of computation, communications and I/O.

Two things to keep in mind are that you must execute on a Lustre (parallel I/O) filesystem and that you need to save your object files and original executable file.

3 Feedback Requested

We would like to request your feedback as part of this assessment.

3.1 Experience Running Your Own Application

- Please describe any difficulty working with your own application that was different from the example
- Please describe what worked well and what didn't work
- In your judgment, will this feature save you time or effort?
- How would you characterize the savings (fewer iterations, less data to examine, etc)?
- What would you estimate for the savings time?

4 Contact information

Don Mason
dmm@cray.com

Margaret Cahir
n13671@cray.com